(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 August 2001 (23.08.2001)

PCT

(10) International Publication Number
WO 01/61466 A1

(51) International Patent Classification[7]: G06F 7/00

(21) International Application Number: PCT/US01/04872

(22) International Filing Date: 16 February 2001 (16.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/505,467 16 February 2000 (16.02.2000) US

(71) Applicant: GOAMERICA, INC. [US/US]; 401 Hackensack Avenue, Hackensack, NJ 07601 (US).

(72) Inventors: WARNOCK, Kevin, L.; 640 Mason Street, #605, San Francisco, CA 94108 (US). WU, John, Shih-Jen; 400 Spear Street, #110, San Francisco, CA 94105 (US).

(74) Agent: MARINA, James, E.; Winston & Strawn, 200 Park Avenue, New York, NY 10166 (US).

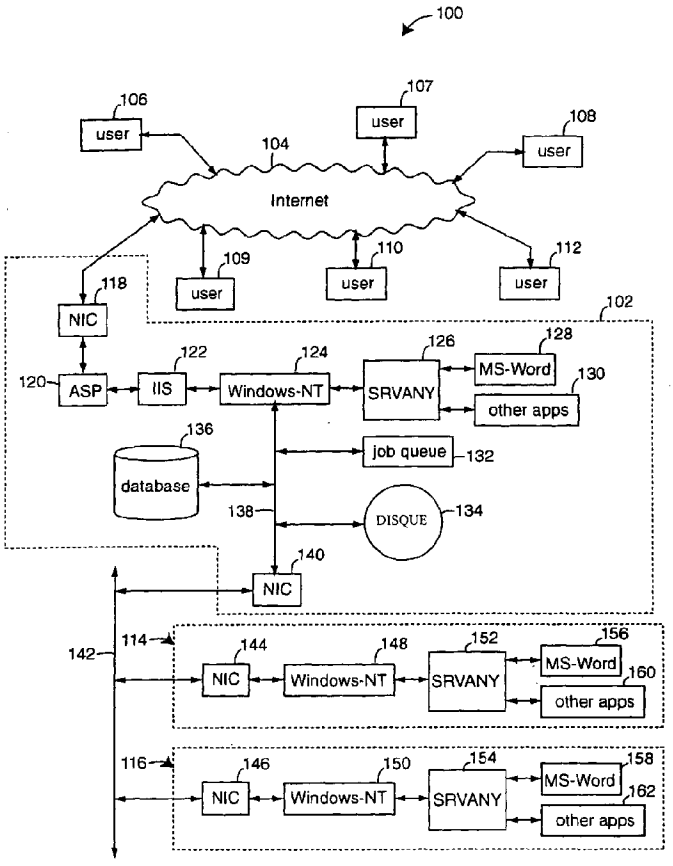
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: DOCUMENT CREATION AND SCHEDULING OF APPLICATIONS' JOBS



(57) Abstract: A document creation system (100) for providing an Internet service in which users (106-112) with browsers at remote locations can look for an appropriate document and format is provided. Such user fills-in-the-blanks and is returned a custom electronic document that can be printed or forwarded to another recipient. A document-creation webserver (102) attends to Internet browsers who log-on and look for a product. Such users (106-112) are qualified and handed-off to a job-master webserver (102). The hand-off provides metadata that was collected from the user (106-112), and schedules the job for the next available document processor. The jobs and metadata are stored in a database (136). Master documents are stored on disk (134). The document processor assigned to do the job collects the metadata from the database using a pointer provided in a job queue (132), and fetches a copy of the appropriate master document. The blanks in the master document copy are filled in using the metadata and/or other data, perhaps from a database (136), and the completed document is returned to the customer over the Internet (104).

WO 01/61466 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DOCUMENT CREATION AND SCHEDULING OF APPLICATIONS' JOBS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to Internet equipment and services, and more particularly to remotely creating and returning completed documents to Internet users who have filled out answers to various questions on a job-input form, and to the automatic generation of mail-merge type wordprocessor documents with fill-in-the-blanks boxes, and more particularly to simplified form generation for Internet document creation webservers.

Description of Related Art

It often happens that people and companies have to write the same basic kinds of letters and fill out the same kinds of forms over and over. Some word processing programs and slide presentation programs provide templates for common formats, e.g., business letters, personal letters, etc. People and companies that have to write some kinds of letters only once or twice often have to write them the way that others have. Hallmark, and other greeting card companies have made quite a business out of eloquently expressing greetings, congratulations, and condolences better than the average person. So preprinted cards with appropriate pictures and a message are bought by millions, and then signed and sent along to the recipient.

Businesses typically have many kinds of form letters that they routinely compose and mail to customers, suppliers, government agencies, and other co-workers. In such situations, it would be advantageous if company policies and legal requirements could be consistently adhered to.

The Judicial Council of California promulgates various court filing forms that are recommended or required for certain California Court filings. The latest revisions and correct use can be difficult for lay people and even small law firms to keep up with.

Electronic forms on disk are available, but these are expensive and become obsolete quickly if subscription upgrades are not also purchased.

The Internet now presents a new media by which even one-time users can take advantage of the highest quality and most up-to-date forms and letters for a very wide spectrum of purposes.

The "boilerplate", graphics, and other artifacts that reoccur in every document in a category need to be designed. Any fill-in-the-blanks dialog boxes that end users fill in to customize a final document must also be designed. Hypertext mark-up language (HTML) code is used to write the program source code that will create the webpage with the correct "boilerplate", graphics, artifacts, and fill-in-the-blanks dialog boxes. But such HTML is difficult to work with, and lay people have no training in the use or writing of HTML.

SUMMARY OF THE INVENTION

In one aspect, an object of the present invention is to provide a document-creation system that can be accessed by many users over the Internet.

Another object of the present invention is to provide a document creation system that allows users to simply and quickly choose the appropriate document format and then supply the information needed to "fill-in the blanks".

A further object of the present invention is to provide an Internet document creation system that is readily scaleable.

A still further object of the present invention is to provide an Internet document creation system that is assembled from common hardware and software components.

Briefly, an Internet document creation system embodiment of the present invention comprises a website that receives information from a user needed to fill out a standardized form document, and then generates a completed ready-to-use document that is sent to the user or the user's intended recipients. Such information may also be stored ahead of time in a database, so the user doesn't need to supply some answers while

on-line. The website comprises a computer platform that interfaces to the Internet with Microsoft Internet information server (IIS) running under a Windows-NT operating system. Application programs like Microsoft WORD97 are made to operate as an NT-service by interfacing it with Microsoft SRVANY to the Windows-NT.

In a second aspect, another object of the present invention to provide a forms-generation system that can be used by lay persons.

Another object of the present invention is to provide a forms-generation system that produces HTML program source code for webpages with correct "boilerplate", graphics, artifacts, and fill-in-the-blanks dialog boxes.

Briefly, a forms-generation system embodiment of the present invention comprises a group of active server page files that run on a WINDOWS-NT platform with IIS and ASP. Each ASP file presents a graphical user interface that allows a lay-user to create mail-merge type wordprocessor documents. Individual ASP files write appropriate HTML phrases in response to a user clicking buttons, entering text, and indicating various preferences.

An advantage of the present invention is that a document-creation system is provided that is quick and expandable.

Another advantage of the present invention is that more document-creation processors can be added as modular units to a base system.

A still further advantage of the present invention is that a document creation system is provided that allows users to simply and quickly choose an appropriate document format, and then supply the information needed to "fill-in the blanks".

A still further advantage of the present invention is that a document-generator is provided that allows for rapid development and implementation of automated website documents.

A still further advantage of the present invention is that a document-generator is provided that minimizes user exposure to programming or scripting languages through an easy-to-use web interface.

A still further advantage of the present invention is that a document generation system is provided that allows content programmers to create the necessary components for automating documents, database tables, and HTML/ASP document forms without needing SQL, HTML, or ASP programming skills or experience. Such can also be used to generate any merge fields needed in the creation of the WORD-file.

The above and still further objects, features, and advantages of the present invention will become apparent upon consideration of the following detailed description of specific embodiments thereof, especially when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a document-creation system embodiment of the present invention;

Fig. 2 is a flowchart for a default.asp embodiment of the present invention;

Fig. 3 is a flowchart for a document.asp embodiment of the present invention;

Fig. 4 is a flowchart for a question.asp embodiment of the present invention;

Fig. 5 is a flowchart for a first half of a results.asp embodiment of the present invention;

Fig. 6 is a flowchart for a second half of a results.asp embodiment of the present invention; and

Fig. 7 is a flowchart for a write_HTML_for_document_form program subroutine embodiment of the present invention that is included in the second half of a results.asp of Fig. 6.

DETAILED DESCRIPTION OF THE INVENTION

INTERNET DOCUMENT CREATION SYSTEM

Fig. 1 illustrates an Internet document creation system embodiment of the present invention, referred to herein by the reference numeral 100. The system 100 comprises a webserver 102 connected to the Internet 104 and many clients or users 106-112. A pair of system-expansion units 114 and 116 modularly increase the ability of webserver 102 to create documents at the requests of users 106-112.

The webserver 102 interfaces to the Internet 104 with a network interface controller 118. An interactive webpage user interface 120 provides functionality similar to Microsoft Corporation (Redmond, WA) Active Server Pages (ASP). An Internet file and application server 122, such as Microsoft Internet Information Server (IIS), runs under control of an operating system 124. Microsoft Windows-NT server is preferably used for the operating system 124. A utility 126 is used to present a wordprocessor program 128 as a service that is available under Windows-NT. For example, SRVANY, as provided by Microsoft as file srvany.exe in the NT-Resource Kit, can be used for the utility 126. Microsoft WORD-97 can be used for the wordprocessor program 128. The SRVANY utility 126 may also be used to run other application programs, e.g., application 130, as Windows-NT services.

Microsoft Windows-NT services are processes that run without requiring a user to be logged on to the system, e.g., Microsoft SQL Server, Microsoft Exchange Server, Dynamic Host Interface Protocol (DHCP), and even Windows Internet Name Service (WINS) servers.

According to Microsoft, the srvany.exe utility allows applications to be run as windows-NT services. This allows applications to survive logoff/logon sequences, and save the re-starting overheads for each new user. Server applications can come-up and service requests even when no user is logged-on. Applications can run and do a task in a specific logon account, different from the currently logged-on user. This ability to run tasks under a specified logon account expands the scope of what can be accomplished with Word 97 and other word processing software programs. For example, it is much more practical and easy to create Adobe PDF files when running under a named account rather than the so-called Local System account that services run as by default. Some

applications may terminate upon logoff, even though they were started as a service, if they don't ignore the wm_endsession message, or ctrl_logoff_event.

A job queue 132 is periodically consulted by the wordprocessor program 128 and other application program 130 to see if there are any jobs pending. A so-called "macro" program can be written with the MS-Word application that will do such job queue checking, e.g. as in Table I. Such macros may be written using Microsoft Visual Basic for Applications. Macro functionality may also be achieved by writing Dynamic Link Libraries (DLLs) that may be invoked from Word. DLLs typically run faster than regular macros. If no job is pending, the macro-program is preferably written to call an API-system function that will put the wordprocessor program 128 or other application program 130 to sleep for a short time. Prototypes that have been built indicate that a sleep-time of one hundred milliseconds produces good results. When the API-system function allows the wordprocessor program 128 or other application program 130 to wake-up, the job queue 132 is checked again. Note that it is not a requirement for the word processing program to be "put to sleep," however this sleep period reduces network traffic and improves performance in the aggregate.

TABLE I

Word Macro

```
' copyright (c) 1999 by Kevin L. Warnock

Declare Function Beep& Lib "kerne132" (ByVal dwFreq As Long, ByVal dwDuration
As Long)
Declare Sub Sleep Lib "kerne132.d11" (ByVal dwMilliseconds As Long)


Sub hot()
'
'hot Macro
'Macro created 01/30/99 by Kevin L. Warnock
`This macro is started by automatically when NT boots, provided Word 97 is set up with
SRVANY.EXE, a utility that ships with the NT Resource Kit from Microsoft. This macro
uses a COM object (Hotworker.Cluster) to determine if there is work to be done. If there
is work, the DraftHot function is called, otherwise, this macro goes to sleep via the
WIn32 api call "sleep." During this time, essentially no CPU time is used.

On Error GoTo err_hot

    Dim ret, i, strSwitch, status, InvokeWord, WorkPath, DelimitedData
    Dim objCluster As Object

    'Set value to 1 so that loop will run at least once.
    strSwitch = "1"

      'Create Hotpaper Cluster object.
      Set objCluster = CreateObject("Hotworker.Cluster")

Do While strSwitch="1"

'Create document by calling method.
ret=objCluster.ReadFromCluster("Compaq NT Server")

Select Case ret

Case "draft"

'Draft document.
status=drafthot(objCluster.MasterDoc, objCluster.ResultDoc,
objCluster.ResultDOcIC, objCluster.DSN, objCluster.WorkPath,
objCluster.Field1, objCluster.Field2, objCluster.Field3, objCluster.Field4)
```

```
'Now implement 2nd phase of two phase commit.
ret = objCluster.StatusCluster(objCluster.WorkKey)

Beep 1500, 100

DoEvents

Case "idle"
  Sleep 100
  DoEvents
End Select

'See if macro should continue by checking switch text file on disk.
'1 = run, 0 = stop.
Open "c:\temp\s\s.txt" For Input As #1
Input #l, strSwitch
Close #1


Loop

  'Release object.
  Set objCluster = Nothing

err_hot:

  Debug.Print "Error: " & Err.Description & " Number: " & Err.Number

End Sub

Function drafthot(MasterDoc, ResultDoc, ResultDOCID, DSN, SQL, WorkPath, Field 1,
Field2, Field3, Field4)
        `This macro actually creates the Word 97 document.

On Error GoTo Err_drafthot

Dim strMsg, DSNstring, PhysicalPath, SaveFormat, ReturnValue

'Change directory to where masters are kept.
ChangeFileOpenDirectory WorkPath

strMsg = strMsg & "3. Changed directory to where masters are kept. Time: "& Now
& vbCrLf

'Open master Microsoft Word 97 document without text datasource attached.
'Note that users may not have any System DSN text drivers installed on the NT Server
'if users want to dynamically attach a text datasource. If users do, a dialog will pop
'up asking users to select which driver to use. This dialog will hang Word and break
this app.
  'In this case, just attach the datasource to the master Word document and save it.  That
will
  'work fine, but users lose the ability to change the application location since the
```

```
datasource
  'is hardcoded into the master document.
  Documents.Open FileName:=MasterDoc, ReadOnly:=True

'Attach SQL Server datasource.
ActiveDocument.MailMerge.OpenDataSource name: _
"", ConfirmConversions:=False, _
Read Only: =False, LinkToSource:=True, AddToRecentFiles:=False, _
PasswordDocument:="", PasswordTemplate:="", WritePasswordDocument:="", _
WritePasswordTemplate:="", Revert:=False, Format:=wdOpenFormatAuto, _
Connection:="DSN=" & DSN & ";", SQLStatement:=SQL, SQLStatementl:=""

'Capture physical path where we will later save file.
PhysicalPath = ActiveDocument.MailMerge.DataSource.DataFields("PhysicalPath_")

'Capture format user requested the file be saved as.
SaveFormat = ActiveDocument.MailMerge.DataSource.DataFields("Format")

  strMsg = strMsg & "4. Opened Word mail merge master document. Time: " & Now &
vbCrLf

  'Perform the merge.
  With ActiveDocument.MailMerge
   .Destination = wdSendToNewDocument
   .Execute
End With

'Put identification on document so we can tell what server it was created on.
Selection. EndKey Unit:=wdStory
Selection.TypeText Text:="Created on Dell 2300/450 server "
Selection.InsertDateTime DateTimeFormat:="M/d/yy h:mm:ss am/pm", _
  InsertAsField:=False
Selection.TypeText Text:="."

strMsg = strMsg & "5. Performed mail merge. Time: " & Now & vbCrLf

'ChangeFileOpenDirectory PhysicalPath & "\"

ChangeFileOpenDirectory "\\hot\hotc\hotdrafts\"
'ChangeFileOpenDirectory "d:\hotpaperenginetestdrafts\"
```

```
    strMsg = strMsg & "6. Deleted any existing version of Word document about to be
saved. Time: " & Now & vbCrLf

    'Save document.
    Select Case SaveFormat
    Case "word97"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".doc") <> "" Then Kill (ResultDoc &
ResultDocID & ".doc")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".doc",
FileFormat:=wdFormatDocument
    Case "word95"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".doc") <> "" Then Kill (ResultDoc &
ResultDocID & ".doc")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".doc",
FileFormat:=11
    Case "works4"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".wps") <> "" Then Kill (ResultDoc &
ResultDocID & ".wps")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".wps",
FileFormat:=17
    Case "word51mac"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".mcw") <> "" Then Kill (ResultDoc &
ResultDocID & ".mcw")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".mcw",
FileFormat:=14
    Case "rtf"
    'Delete any existing version first.
    If Dir(ResultDoc & ResultDocID & ".rtf") <> "" Then Kill (ResultDoc &
ResultDocID & ".rtf")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".rtf",
FileFormat:=wdFormatRTF
    Case "wp51win"
    'Delete any existing version first.
    If Dir(ResultDoc & ResultDocID & ".doc") <> "" Then Kill (ResultDoc &
ResultDocID & ".doc")
    'Save file.
    ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".doc",
FileFormat:=22
```

```
    Case "wp51dos"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".doc") <> "" Then Kill (ResultDoc &
ResultDocID & ".doc")
        'Save file.
        ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".doc",
FileFormat:=21
    Case "html"
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".html") <> "" Then Kill (ResultDoc &
ResultDocID & ".html")
    'Save file.
    ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".html",
FileFormat:=10
    Case Else
        'Delete any existing version first.
        If Dir(ResultDoc & ResultDocID & ".doc") <> "" Then Kill (ResultDoc &
ResultDocID & ".doc")
    'Save file.
    ActiveDocument.SaveAs FileName:=ResultDoc & ResultDocID & ".doc",
FileFormat:=wdFormatDocument
    End Select

'Dim conv
'For Each conv In FileConverters
'    If conv.CanSave Then Debug.Print conv.FormatName & conv.SaveFormat
'Next conv

    strMsg = strMsg & "7.  Saved completed Word mail merge document. Time: " &
Now & vbCrLf

    strMsg = strMsg & "8. Wrote out HTML link screen. Time: " & Now & vbCrLf

    'Set return value of function.
    ReturnValue = "Word 97 document drafted successfully."
    drafthot = ReturnValue

    strMsg = strMsg & "9. Set Return Value of function. Time: " & Now & vbCrLf

Exit_drafthot:

    'Close all documents without saving.
    'If Documents.Count > 0 Then Documents.Close
SaveChanges:=wdDoNotSaveChanges
```

```
    ActiveWindow.Close SaveChanges:=wdDoNotSaveChanges
    ActiveWindow.Close SaveChanges:=wdDoNotSaveChanges

strMsg = strMsg & "10. Closed open documents. Time: " & Now & vbCrLf

    'Write return value to file. This value will be returned to ASP script and will contain
word 'error' if something went wrong.
    'Open WorkPath & "\WWWizWordReturnValue.txt" For Output Access Write Lock
Write As #2
    'Print #2, ReturnValue
    'Close #2

strMsg = strMsg & "11. Wrote return value to disk. Time: " & Now & vbCrLf

    'Write progress string to file.
    'Open WorkPath & "\WWWizWordProgress.txt" For Output Access Write Lock Write
As #2
    'Print #2, strMsg
    'Close #2

    Debug.Print ReturnValue

    Exit Function

Err_drafthot:

    'Set return value of function.
    RetumValue = "ERROR: Word 97 document not drafted. Error Description: " &
Err.Description & "; Error number: " & Err.Number & "; Time: " & Now

    'Write out ReturnValue to debug window.
    Debug.Print ReturnValue

    strMsg = strMsg & "ERROR HANDLER: " & ReturnValue & vbCrLf

    'Exit this function.
    GoTo Exit_drafthot

End Function
```

The job queue 132 can be written as a Visual Basic program, e.g., as list in Table II. Jobs are posted into a pending list or queue. Any wordprocessor program 128 or other application program 130 that is ready to do a job looks at the pending list. If an appropriate job is recognized, its entry in the job queue 132 is moved into an "in-progress" list. The wordprocessor program 128 or other application program 130 then

pulls informational pieces it needs from a computer data storage disk 134 and a database 136. The information pieces can also be retrieved in an alternative manner, for example, from a memory location. The job is then executed, and the results in the form of an electronic document are deposited in the disk 134. The Windows-NT 124 then forwards such document to the user 106-112 or the user's designated recipients. When the job is finished, its in-progress entry in the job queue 132 is moved into a "done" list.

TABLE II

(VB Queue)

```
' copyright (c) 1999 by Kevin L. Warnock

Private strMasterDoc As Variant
Private strSQL As Variant
Private strFieldl As Variant
Private strField2 As Variant
Private strField3 As Variant
Private strField4 As Variant
Private strDSN As Variant
Private strMacro As Variant
Private strSubmitTime As Variant
Private strResultDoc As Variant
Private strResultDocID As Variant
Private strWordVisible As Variant
Private strMyname As String
Private strWorkpath As String
```

```
Public Function ReadFromCluster(MachineName)

On Error GoTo Err_ReadFromCluster

    'Create variable for central watch directory for work coming in from IIS.
    HotqueuePath = "\\hot\hotc\temp\q\"

    'Create variable for directory for documents that are being currently processed by a
Word cluster machine.
    HotqueueDrafting = "\\hot\hotc\temp\d\"

    'Create variable for directory where completely finished requests are sent.
    HotqueuePathFinished = "\\hot\hotc\temp\f\"

    'Look to see if there are any requests waiting to be processed.
    mypath = HotqueuePath & "*.txt"
    Myname = Dir(mypath)
    strMyname = Myname

    'If there is a request, move it to the "in process" directory and read values into property
variables so Word cluster processes can access values.
    If Myname <> "" Then

    'To move files, we use filesystemobject from ASP. Cluster machines need ASP so they
can use this command?
    Set fso = CreateObject("scripting.filesystemobject")
    Set a = fso.getfile(HotqueuePath & Myname)
    a.Move HotqueueDrafting

    'Read contents of request and assign values to property values.
    Open HotqueueDrafting & Myname For Input As #1
    Input #l, strMacroTemplate, strMacroName, strMasterDoc, strResultDoc,
strResultDocID, strDSN, strSQL, strInvokeWord, strDelimitedData, strHeader,
strRecord,
strWorkpath, strWordVisible, strFieldl, strField2, strField3, strField4
    Close #1

    'Close file system object.
    Set fso = Nothing

    'Assign return value for this function.
    ReadFromCluster = "draft"

Else

'Assign return value for this function.
```

```
    ReadFromCluster = "idle"

End If

Exi_ ReadFromCluster:

    Exit Function

Err_ReadFromCluster:

    ReadFromCluster = "Error number: " & Err.Number & ", error description: " &
Err.Description & ", error source: " & Err.Source & "."

    GoTo Exit_ReadFrom Cluster

End Function

Public Function StatusCluster(WKey)
    'After document is drafted, we want to move request
    'file from the HotqueueDrafting directory to the
    'HotqueuePathFinished directory, since it will be nice
    'to be able to examine the in process drafts in
    'Windows explorer. Upon successful drafting, the
    'request files can be moved to the 'finished' directory
    'to keep a record of what was done.

'Pass in WKey, which is the filename of the request
'file to move.

On Error GoTo Err_StatusCluster

    'Create variable for directory for documents that are being currently processed by a
Word cluster machine.
    HotqueueDrafting = "\\hot\hotc\temp\d\"

    'Create variable for directory where completely finished requests are sent.
    HotqueuePathFinished = "\\hot\hotc\temp\f\"

    'Look to see if there are any requests waiting to be processed.
    mypath = HotqueueDrafting & WKey
    Myname = Dir(mypath)
```

```
    'If there is a request, move it to HotqueueDraftng directory and read values into
property variables so Word cluster processes can access values.
    If Myname <> "" Then

    'To move files, we use filesystemobject from ASP. Cluster machines need ASP so
they can use this command?
    Set fso = CreateObject("scripting.filesystemobject")
    Set a = fso.getfile(HotqueueDrafting & Myname)
    a.Move HotqueuePathFinished
    Set fso = Nothing

    'Assign return value for this function.
    StatusCluster = "success"

Else

    'Assign return value for this function.
    StatusCluster = "failure"

End If

Exit_StatusCluster:

    Exit Function

Err_StatusCluster:

    StatusCluster = "Error number: " & Err.Number & ", error description: " &
Err.Description & ", error source: " & Err.Source & "."

    GoTo Exit_StatusCluster
End Function

Public Property Get MasterDoc() As String
    MasterDoc = strMasterDoc
End Property
Public Property Get SQL() As String
SQL = strSQL
End Property
Public Property Get Macro() As String
    Macro = strMacro
End Property
Public Property Get SubmitTime() As String
    SubmitTime = strSubmitTime
End Property
```

```
Public Property Get ResultDoc() As String
   ResultDoc = strResultDoc
End Property
Public Property Get ResultDocID() As String
   ResultDocID = strResultDocID
End Property
Public Property Get WordVisible() As String
   WordVisible = strWordVisible
End Property
Public Property Get Field 1() As String
   Field 1 = strFieldl
End Property
Public Property Get Field2() As String
   Field2 = strField2
End Property
Public Property Get Field3() As String
   Field3 = strField3
End Property
Public Property Get Field4() As String
   Field4 = strField4
End Property
Public Property Get DSN() As String
   DSN = strDSN
End Property
Public Property Get WorkKey() As String
WorkKey = strMyname
End Property
Public Property Get WorkPath() As String
   WorkPath = strWorkpath
End Property
```

A "writer" code is invoked from ASP 120, in response to users 106-112, and puts new jobs to do in the job queue 132. A typical source code for this is listed in Table III.

TABLE III

```
' copyright (c) 1999 by Kevin L. Warnock

Public Function SubmitToCluster(MacroTemplate, MacroName, MasterDoc, ResultDoc,
ResultDocID, DSN, SQL, InvokeWord, DelimitedData, Header, Record, WorkPath,
WordVisible, Fieldl, Field2, Field3, Field4)

On Error GoTo Err_SubmitToCluster

   'Write text file to pass parameters to Word macro.
   Open "c:\temp\q\" & ResultDoc & ResultDocID & ".txt" For Output As #1
   Write #1, MacroTemplate, MacroName, MasterDoc, ResultDoc, ResultDocID, DSN,
SQL, InvokeWord, DelimitedData, Header, Record, WorkPath, WordVisible, Field,
Field2, Field3, Field4
   Close #1

   SubmitToCluster = "Success"

Exit_SubmitToCluster:

   Exit Function

Err_SubmitToCluster:

   SubmitToCluster = "Error number: " & Err.Number & ", error description: " &
Err.Description & ", error source: " & Err.Source & "."
   GoTo Exit_SubmitToCluster


End Function
```

A system bus 138 in webserver 102 allows expansion through a local area network (LAN) network interface controller (NIC) 140. If the capabilities of the webserver 102 to create documents for the users 106-112 are insufficient because of too great a user demand, a local area network (LAN) 142 can be added-on. Such LAN 142 allows one or more modular expansion units 114 and 116 to be added. The advantage of this is that no extra capacity beyond that which is really necessary must be purchased and installed. Each of the modular expansion units 114 and 116 will access the job queue 132 and look for work to do. Each of the modular expansion units 114 and 116 will pickup metadata and deposit their work products from and to the database 136 and the disk 134.

The LAN 142 is preferably private and inaccessible to the Internet 104 and any of the users 106-112. Standard ETHERNET components can be used to implement LAN 142, NIC 140, and each of a pair of NIC's 144 and 146.

The modular expansion units 114 and 116 replicate a part of the webserver 102 and provide parallel document-creation processing. Each has a Windows-NT operating system 148 and 150. A pair of SRVANY utilities 152 and 154 allow a corresponding set of MS-Word wordprocessor programs 156 and 158 to run as Windows-NT services in parallel with the MSWord 128. Other applications can also be run in parallel as Windows-NT services, and these are represented in Fig. 1 as other applications 160 and 162.

The webserver 102 has been described herein as being completely implemented with various products marketed by Microsoft Corporation, but other similar products from competing suppliers can be used just as well. For example, the webserver 122 may be from Netscape or Apache, the word processor 128 may be from Corel or Lotus, and the operating system 124 may even be Linux rather than Windows NT server. In this case, only word processors designed for Linux, such as WordPerfect from Corel, could be substituted for the word processor 128. The scripting engine ASP 120 may be from Allaire (Cold Fusion), or may be implemented with Java Server Pages, Java Servlets, Perl or Java, among others.

In operation, the users 106-112 are greeted, checked, and screened by another website or portion of this website before being directed to the webserver 102. Such other website allows the users to find, explore, and subscribe to the document-creation services being offered. Once the user 106-112 has qualified as a customer or other authorized user, a document format is selected for use. Such document format will require the user to supply answers to various questions, including information that will be used to fill-in-the-blanks. The answers to the questions, the document-format selection, and the information to fill in the blanks will be transferred in metadata form to the website 102.

Such metadata represents the specifics of a job that needs doing, e.g., a job order. The particulars of the job order are deposited in the database 136. A pointer to this job order and a code representing what kind of job it is are placed in the new job list in the

job queue 132. A large selection of standardized "master documents" is stored in the disk 134 and for redundancy and speed, on the local disks of the slave Word machines 114 and 116.

When any of the MS-Word programs 128, 156, and 158, or any of the other applications 130, 160, and 162, "wake-up" from their periodic enforced sleep periods, they each independently check the new job list in the job queue 132 for work that needs doing. If a compatible job that needs doing exists, the application moves the pointer from the new job list to the in-progress list. The pointer data is used as an index to find all the metadata related to the new job that is stored in the database 136. An appropriate master document is fetched from the disk 134 or the local disks of the slave machines, if appropriate. The application then creates the purchased or free document, as many documents may be given away to the public, for the user 106-112 and parks it temporarily in the disk 134. From there it is transmitted to a web-address specified by the user 106-122 in the original job order.

Microsoft Internet Information Server (IIS) is an Internet file and application server included with the Microsoft Windows-NT Server operating system. According to Microsoft, IIS can be used alone as a webserver, or in conjunction with compatible technologies to set up Internet commerce, to access and manipulate data from a variety of data sources, and to build Web applications that take advantage of server script and component code to deliver client-server functionality. The latest information is available on Microsoft's IIS website. IIS provides security, networking, and administration functionality, and has built-in capabilities to help administer secure websites, and to develop and deploy server-intensive Web applications. IIS includes a Web-based administration tool that makes remote administration of webservers possible through use of a Web browser. Users thus have the ability to remotely manage their own sites, safe transmission of data is done with secure sockets layer (SSL).

The IIS is tightly integrated with Windows-NT Server, and so it can utilize many of the management tools within Windows-NT Server to help administer the system, e.g., User Manager for Domains, Performance Monitor, Network Monitor, Event Viewer, and Simple Network Management Protocol (SNMP). In IIS 4.0, it is possible to create an unlimited number of websites on a single IP address, and to have different configuration

information for each one. This has been challenging in the past, because each IP address could have only one domain name. The Internet Engineering Task Force's HTTP 1.1 allows multiple domain names on one IP-address by specifying the host header information that gets a user to the right website. IIS comes with the support necessary for those browsers to access sites. When multiple websites are enabled on one machine, IIS 4.0 provides users the ability to manage how much network bandwidth is provided to each website. Such "bandwidth throttling" ensures enough bandwidth is made available to each of the sites on the machine. Sites that publish static HTML pages, e.g., ".htm" files, can take full advantage of bandwidth throttling in IIS.

The Windows-NT 4.0 Option Pack includes Microsoft Site Server Express, so site administrators can analyze server log files and produce reports that detail user behavior, analyzes site content and correct errors, and publishes webpages to local and remote servers.

When secure transactions are required over the Internet, IIS and Windows-NT Server provide support for Secure Sockets Layer 3.0 (SSL), enabling information to be exchanged between clients and servers. SSL 3.0 provides a way for the server to verify who the client is through the use of digital certificates, without requiring a server logon. IIS issues and manages these certificates through Microsoft Certificate Server, and maps them to user accounts on a machine that gives the user the correct level of access to files and services. Windows-NT Server and IIS also support basic authentication, sending of unencrypted user names and passwords, Challenge/Response cryptographic authentication of passwords, and server-gated crypto 128-bit encryption for digital certificates used in transactions with banks and other financial institutions.

Transaction support in Windows-NT Server and IIS is implemented through Microsoft Transaction Server (MTS) 2.0 and tracks the success or failure of complete system processes, such as ordering or accessing and manipulating data. When ASP pages are declared to be transactional, Transaction Server handles the details of creating the defined transactions that occur within the page. Transaction components are activated when needed and deactivated when not in use to save system resources. MTS management is also controlled through the Microsoft Management Console.

An OLE-type control enables developers to create Microsoft Visual Basic applications that will function as Windows-NT services. With the NTService control, users can install a service, log events, and respond to start, stop, pause, and continue events. The operating system interface for services requires a callback function and blocks the main thread of execution. Because Visual Basic 4.0 has neither threads nor callback functions, developers cannot call these application programming interfaces (APIs) directly. Visual Basic applications can rely on the SRVANY utility provided in the Windows-NT Resource Kit which allows any executable to be started by the system at boot time. However, the communication between SRVANY and the application is limited to standard window messages, so such solution is not very robust. Moreover, SRVANY lacks support for the pause and continue functions.

An NTService OLE control could alternatively be used to eliminate the need for a helper processes like SRVANY utility 126 by enabling Visual Basic applications to directly interface with the services' APIs. Such OLE control translates signals from the operating system into events that can be processed by a Visual Basic application. It also provides support for functions that services typically require, such as installation and event logging. Creating a service begins by dropping the NTService control on a form. Services typically do not have any user interface, but in Visual Basic the control needs a form to serve as a container. At a minimum, users will need to set some properties and implement the installation routine and handlers for the Start and Stop events.

Many of the details of creating services in C or C++ are well documented and various examples assume that the entire application is designed around the Windows-NT service framework defined by the Win32 APIs. The NTService control must bridge the services' interfaces with the Visual Basic application model. During initialization, services call the StartServiceCtrlDispatcher API, passing a callback function that is used by the service controller for service initialization. The control cannot call the dispatcher function in a method because this function blocks the caller until the service is stopped. To solve this problem, the control creates a separate thread that is responsible for calling the dispatcher. This allows the primary thread used by Visual Basic to continue processing while the other thread is blocked by the service's dispatcher.

While the application is running as a service, events will fire from the service thread and the control request handler. Both of these functions execute on different thread contexts from the Visual Basic code. Because Visual Basic is not thread-safe, the apartment-model OLE control rules specify that all calls must occur in the same thread where the object was created. The effect on the NTService control is that it must transfer execution to the primary thread in order to generate notification events. This is done by posting window messages to the control's invisible window and letting the window message dispatch loop inside Visual Basic transfer execution to the control.

The control's window is critical to the routing of messages from the service's dispatcher to the Visual Basic application. Because OLE control containers have some discretion as to when the control window is created, the control informs the container that the window must exist always. This is done by marking the control with the OLEMISC_SIMPLEFRAME attribute and enabling the simple frame interface in the control constructor.

Even though an application is running as a service it does not mean that it can handle the processing requirements of application servers such as Microsoft SQL Server or Microsoft Exchange Server. Visual Basic services are single-threaded and do not expose programming interfaces. The Distributed Component Object Model (DCOM) in Windows-NT version 4.0 allows the service to be callable through OLE Automation interfaces locally or from a remote computer.

According to Microsoft, this type of service can be used to augment application servers. For example, if users need to capture a real-time data broadcast in a SQL database, the Windows-NT service support can be added to make the application easier to administer. Likewise, users could write Microsoft Exchange Server mailbox agents in a higher-level language, taking advantage of existing components, such as OLE Messaging, to simplify development.

The NTService control lets developers create Visual Basic applications that install and run as Windows-NT services. These applications benefit from automatic startup, remote administration, and event logging.

Active Server Pages is a programming environment for combining HTML, scripting, and components to create Internet applications that run on the server. If users are already creating websites that combine HTML, scripting, and some reusable components, users can use ASP to glue these items together. Users can create an HTML interface for the application by adding script commands to the HTML pages and users can encapsulate the business logic into reusable components. These components can be called from script or other components.

Microsoft Active Server Pages (ASP) is a server-side scripting environment that users can use to create dynamic webpages or to build Web applications. ASP pages are files that contain HTML tags, text, and script commands. ASP pages can call ActiveX components to perform tasks, such as connecting to a database or performing a business calculation. With ASP, users can add interactive content to the webpages or build entire Web applications that use HTML pages as the interface to the customer.

When ASP is incorporated into a website, a user is allowed to bring up the website where a default page has the extension ".asp". The user's browser requests the ASP file from the webserver. A server-side script begins to run with ASP. The requested file is processed top-down by ASP, executing any script commands contained in the file. An HTML webpage is produced as a result and is sent to the user's browser. Because such script runs on the server, the webserver does all of the processing so standard HTML pages can be generated and sent to the browser. The webpages are limited only by what the webserver supports. Another benefit of having the script reside on the server is that the user cannot "view source" on the original script and code. Instead, the user sees only the generated HTML as well as non-HTML content, such as XML, on the pages that are being viewed.

DOCUMENT-FORMAT GENERATOR

When an end-user accesses a document on website 102, information needed by the document must be entered into a document form. Such form comprises a mix of HTML and ASP code. When any information on the document form is submitted to the

server, the ASP 120 processes the entered data and inserts it into a table in the database 136. These tables are specific to the documents, and include only the specific fields needed by the document for the information.

A document generation process embodiment of the present invention can be initiated by dropping a queue file into a directory which is monitored by one of the wordprocessors 128, 156, 158, etc. The wordprocessor 128, for example, reads job queue 132 to determine which master document in disk 134 to use and where in database 136 the corresponding document data has been stored. It then opens a WORD-file for the master document that includes mail-merge fields for population with the document data. Wordprocessor 128 then opens a connection to a table in the document database 136, and gets the document data for merging into the document. As a last step, the document file is saved.

Document generator embodiments of the present invention generally allow for rapid development and implementation of automated documents for the website 102. Such a wizard tool can minimize user exposure to programming or scripting languages through an easy-to-use web interface. Content programmers can create the necessary components for automating documents, database tables, and even HTML/ASP document forms without needing SQL, HTML, or ASP programming skills or experience. Such can also be used to generate any merge fields needed in the creation of the WORD-file.

Software embodiments of the present invention can be implemented as ASP files made accessible to ASP 120. Each such ASP-file presents a screen or graphical user interface (GUI) to a user. For example, a default.asp allows a content programmer to select a document master for editing, or create a new document. If a new document is created, the computer host will verify that the name conforms to set-naming conventions and will copy question sets from existing document masters.

A documents.asp displays and allows editing of the document master settings including its name, description, location, and other configuration information. The program verifies changes to the document master to ensure information has been entered correctly. The questions asked by the document are listed in this screen. A content programmer preferably can delete, create, or reorder the set-of-questions. Clicking on a

question (in hypertext) preferably allows the content programmer to edit the details of the question. Options can include being able to generate the database table and basic WORD-template with merge fields, and the HTML/ASP document form.

A questions.asp screen preferably allows a content programmer to view or edit question details. Here the content programmer sets the question name, text, type, and default value.

A results.asp screen generates a database table, WORD-template, and HTML/ASP document form. It then displays the results.

Fig. 2 is a flowchart for a default.asp embodiment of the present invention, and is referred to herein by the general reference numeral 200. A step 202 checks if a user has entered and a document has been selected. A subroutine 204 gets a list of document masters from document masters database table, displays a list of document masters on screen, displays a submit button, displays a "new document master name" input field, and displays a list of document masters for copying question sets. If user has selected a document master in a decision 206, a document master is set in a step 208. If a decision 210 detects a "copy question set" is selected, then a step 212 gets the document master name of the question set to copy. These then redirect to a document.asp program (Fig. 3). If a decision 214 detects the user selected "Create New Document Master", then a step 216 gets a new document master name. A decision 218 looks to see if a "copy question set" selected. If so, a step 220 gets the document master name of the question set to copy. A step 222 sets a new-document-master flag, and program control is redirected to document.asp program (Fig. 3).

Fig. 3 is a flowchart for a document.asp embodiment of the present invention, and is referred to herein by the general reference numeral 300. A step 302 checks to see if a new-document-master flag is true. If so, a subroutine 304 sets the document master field default values. It checks the document master name. If the name already exists, control is redirected to default.asp 200 (Fig. 2). If the name does not conform to convention, again control is redirected to default.asp 200 (Fig. 2). Subroutine 304 inserts any new document master data into a document masters database table. If a copy-question-set is true, question-set-information is copied to a document master questions database table. It

then selects the document master. A decision 306 looks to see if the existing document master has been selected. If so, a subroutine 308 retrieves document master information from document masters table, and then gets document master questions from the document master questions table. It displays (a) document master fields with document data, (b) an update information button, (c) a questions table with question links, (d) a new-question button, (e) a checkbox for each of "generate-database-table" and "HTML/ASP document form", and (f) a "submit-generate" button. A decision 310 checks to see if a "new-question" button has been pressed. If so, a step 312 sets a new-question flag. Program control is then redirected to question.asp (Fig. 4). A decision 314 looks if an "update-information" button has been pressed. If so, a subroutine 316 verifies form data. If such form data has problems, an error message is displayed. If the form data is OK, the program continues, e.g., to update document master data in a document masters table. A decision 318 looks if generate button pressed, redirect to results.asp (Fig. 5).

Fig. 4 is a flowchart for a question.asp embodiment of the present invention, and is referred to herein by the general reference numeral 400. A decision 402 checks if a new-question flag has been set. A step 404 sets any default-question field values. A decision 406 checks if a question has been selected. A step 408 gets question data from a document-master questions table. A decision 410 checks if an "update-question" button has been pressed. If so, a step 412 updates or inserts question data into a document master questions table. A subroutine 414 displays the form, e.g., question fields with field name, question text, question type, question parameters, and default value. It also displays an "update-question" button.

Fig. 5 is a flowchart for a first part of a results.asp embodiment of the present invention, and is referred to herein by the general reference numeral 500. A decision 502 checks if a generate database table button has been selected. If so, a step 504 gets a list of questions for document master from document master questions table. A subroutine 506 builds an SQL-type command line that will create a table. The command string start is built with a SQL create table command and a name table with document master name. A program loop iterates through a question list. For each question a field name is added to the SQL command to create a new column for the field. The question types are looked

up. A database field type and length is fetched, and appended to the SQL command. The SQL command is closed. And the SQL command is executed to create the table. A decision 508 checks if the "generate HTML/asp document form" has been selected. If so, control passes to a generate HTML/ASP document program (Fig. 6). Otherwise, control returns.

Fig. 6 is a flowchart for a second part of a results.asp embodiment of the present invention, and is referred to herein by the general reference numeral 600. A step 602 opens text to write named after a document master. A step 604 write file includes to text file. A subroutine 606 writes variable definitions. It gets a list of questions for document master from document master question table for each question. Then it writes a variable definition using field name. A step 608 writes a main subroutine to text file, this subroutine controls the overall flow of the code. A step 610 writes a submit_doc_values subroutine that will insert any document form values into the database. It then writes commands to update a documents table which tracks the document created by each user. It also writes commands to update a document specific table that holds any data entered into the document form. Step 610 gets a list of questions for the document master from document master question table. For each question, a variable cleanup routine is run which removes illegal input, e.g., input that is too long is truncated. Also for each question, the variable is added to an SQL statement that inserts data into a document-specific table. A step 612 writes a get_doc_values subroutine that retrieves and populates form fields with past values. Such is used for a "reused past answer set" feature. Commands are written that check if a document ID was passed to the document form. If not, the rest of subroutine 612 is skipped. Commands are then written automatically to retrieve past answers using document ID from document specific table. Document values are assigned to the form-variables. A list of questions for document master is retrieved from the document master question table. Then for each question, a document value assignment is written to the form-variable. A get-default-values subroutine 614 writes a routine that assigns document variables default values if no values have been assigned. A list of questions is retrieved for the document master from a document master question table along with any default values. For each question, a default value assignment is written to form the variable. A write HTML form code subroutine 616 gets document master information from document masters table. It writes

HTML header information including title and keywords. Any document description and instructions are written. Sponsorship banners from documents under license and companies table are retrieved for use. The HTML-code for including the sponsorship banners is written. It writes the HTML for the document form. Fig. 7 details this last step more fully. An HTML phrase for a "look and feel" template is also automatically inserted into the HTML phrase.

Fig. 7 is a flowchart for the "writes the HTML for the document form" part of the results.asp 600 (Fig. 6). A step 702 creates a table, and gets a list of questions for document master from document master question table. For each question, it creates a row, writes question text, and writes an input field HTML. An end table mark is made. A check is made to see if the document master has a disclaimer statement, and if not, writes the necessary HTML to include an appropriate legal disclaimer. The last part of subroutine 702 writes the HTML-code needed to support a "submit and rest" button.

Although particular embodiments of the present invention have been described and illustrated, such is not intended to limit the invention. Modifications and changes will no doubt become apparent to those skilled in the art, and it is intended that the invention only be limited by the scope of the appended claims.

THE INVENTION CLAIMED IS

1. A document-creation system, comprising:

a webserver for interfacing to the Internet and a plurality of remote network users and browsers, and including a network operating system that can call a variety of server processes, and an active server page process that provides interactive webpages for each of said remote network users and browsers;

a database included in the website and for receiving metadata from said plurality of remote network users and browsers needed to create a plurality of user documents;

a disk included in the website and including a selection of master documents needed to create said plurality of user documents;

at least one application program for merging a particular piece of said metadata and a selected one of said master documents; and

a job queue for providing scheduling of the application programs according to said metadata and selected ones of said master documents;

wherein, the application programs create said plurality of user documents from copies of said master documents and metadata and a product is returned to corresponding ones of said plurality of remote network users and browsers.

2. The document-creation system of claim 1, wherein:

the job queue and application programs are such that each application program periodically and independently checks the job queue for work.

3. The document-creation system of claim 1, wherein:

the job queue and application programs are such that each application program puts itself to sleep for a period of time if the job queue has no work.

4. The document-creation system of claim 1, further comprising:

a utility for providing an interface between said network operating system and the application programs that allow the application programs to run as server processes.

5. The document-creation system of claim 1, further comprising:

a local area network (LAN) connected to the network operating system and for providing a backend expansion backbone for additional application programs.

6. The document-creation system of claim 5, wherein:

the LAN is not directly accessible from the Internet.

7. The document-creation system of claim 1, wherein:

at least one of the application programs is a wordprocessor program that includes a macro capability and macro instructions that puts the application program to sleep for a period of time if the job queue has no work.

8. The document-creation system of claim 1, wherein:

the job queue is implemented with a VISUAL BASIC program and includes a job-waiting list, a job-in-progress list, and a job-done list.

9. The document-creation system of claim 8, wherein:

the webserver further includes an Internet information server connected to said active server page process that supplies said metadata to the database and a pointer to said metadata in the database that is placed in said job-waiting list.

10. The document-creation system of claim 9, wherein:

said VISUAL BASIC program causes said pointer to be moved from said job-waiting list to said job-in-progress list when of one of the application programs has been assigned a corresponding job.

11. The document-creation system of claim 9, wherein:

said VISUAL BASIC program causes said pointer to be moved from said job-in-progress list to said job-done list when said one of the application programs has completed a corresponding job.

12. A document-generating wizard, comprising:

a server-side scripting environment for providing dynamic webpage creation;

a plurality of active server pages that each include HTML tags, text, and script commands for the server-side scripting environment;

a database that includes a plurality of document templates;

an HTML-code generator that automates a writing of HTML-code to be embedded in any of said plurality of document templates;

a graphical user interface connected to the server-side scripting environment, the HTML-code generator, and the database;

wherein, a user directly provides no HTML-code for inclusion in any document for execution by the server-side scripting environment.

13. The wizard of claim 12, wherein:

the server-side scripting environment includes MICROSOFT ACTIVE SERVER PAGES; and

the active server pages can call an ACTIVEX component to connect to the database and do business calculations;

wherein, said users can add interactive content to a webpages and build entire Web applications that use HTML pages as a customer interface.

14. The wizard of claim 12, wherein:

the server-side scripting environment allows a user with an Internet browser to bring up a website with a default.asp page, and that can request any asp-type files from a webserver to make a server-side script execute.

15. The wizard of claim 12, wherein:

the server-side scripting environment is such that a requested asp-type file is processed top-down to execute any script commands contained in said file;

wherein, an HTML webpage is dynamically produced as a result and is sent to a user's Internet browser by a webserver that does about all of the processing; and

wherein, because said script commands reside on a webserver, said user cannot view any original script and code source, said user sees only a generated-HTML on said HTML webpage being viewed.

16. The wizard of claim 12, further comprising:

a plurality of ASP-type files associated with the server-side scripting environment for creating the graphical user interface; and

a default.asp file included in the plurality of ASP-type files that enables a content programmer to select a document master for editing, or create a new document.

17. The wizard of claim 12, further comprising:

a plurality of ASP-type files associated with the server-side scripting environment for creating the graphical user interface; and

a documents.asp included in the plurality of ASP-type files that displays and allows editing of any document-master settings, and that can verify any changes to a document master to ensure information has been entered correctly;

wherein, a list of questions asked by said document master are listed in a screen display such that a content programmer can delete, create, and reorder a set-of-questions.

18. The wizard of claim 12, further comprising:

a plurality of ASP-type files associated with the server-side scripting environment for creating the graphical user interface; and

a questions.asp file included in the plurality of ASP-type files that allows a content programmer to view and edit document question details that include a question name, any text strings, a type, and a default value.

19. The wizard of claim 12 further comprising:

a plurality of ASP-type files associated with the server-side scripting environment for creating the graphical user interface; and

a results.asp file included in the plurality of ASP-type files that generates a database table, a WORD-template, and an HTML/ASP document form, and that causes such to be displayed on-screen to a user.

20. The wizard of claim 19, wherein:

the results.asp files produces a resulting document that queries a third-party for information that must be entered into a document form which comprises a mix of HTML and ASP code, and wherein said information is later submitted to a server for processing any entered data and inserts such into a table in the database.

21. A document generator for rapid development and implementation of automated documents for a website, comprising:

a webserver for connection to the Internet;

an active server pages (ASP) server-side scripting environment hosted on the webserver;

a plurality of ASP-type files associated with the server-side scripting environment for creating the graphical user interface;

a default.asp file included in the plurality of ASP-type files that enables a content programmer to select a document master for editing, or create a new document;

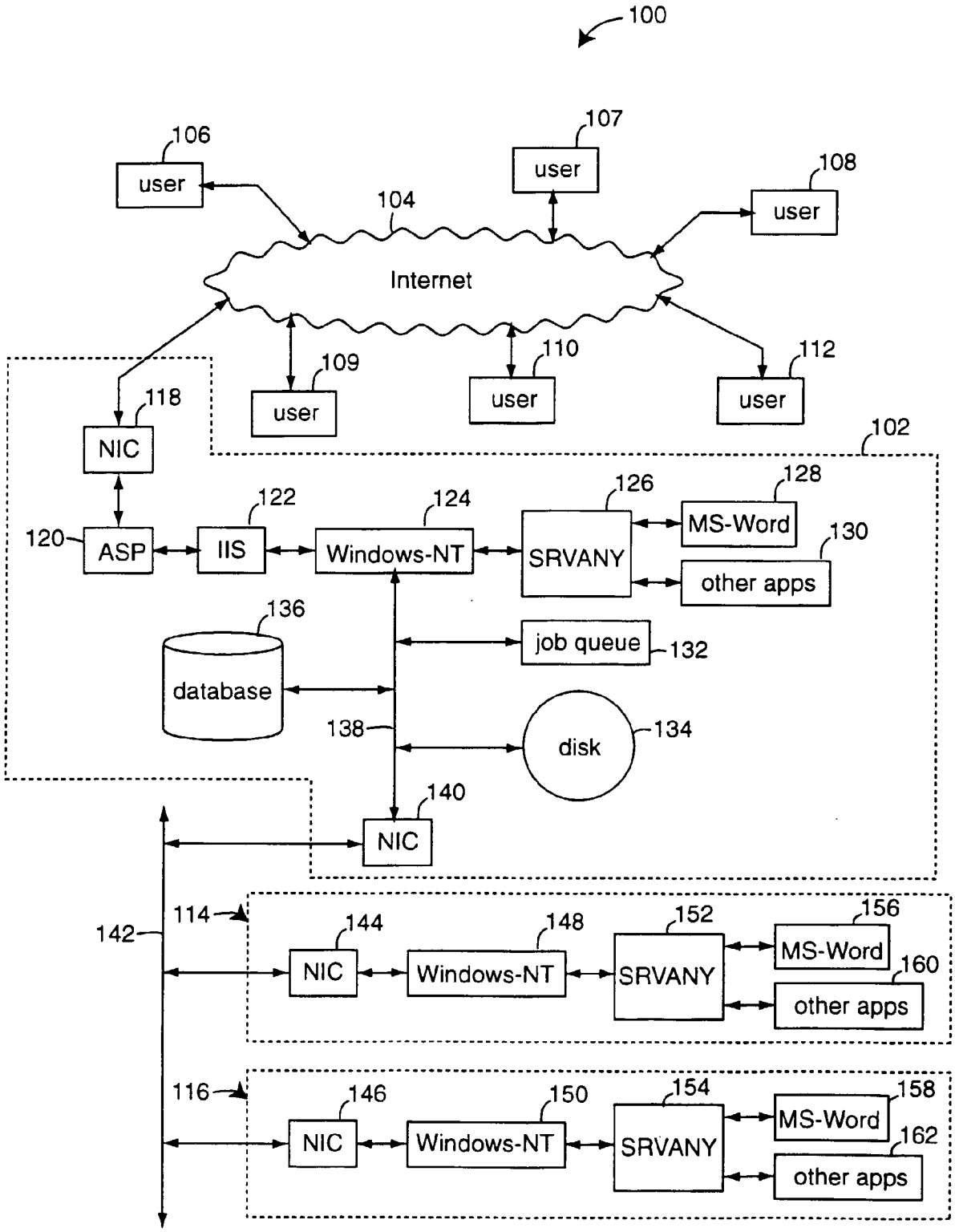
a documents.asp included in the plurality of ASP-type files that displays and allows editing of any document-master settings, and that can verify any changes to a document master to ensure information has been entered correctly;

a questions.asp file included in the plurality of ASP-type files that allows a content programmer to view and edit document question details that include a question name, any text strings, a type, and a default value; and

a results.asp file included in the plurality of ASP-type files that generates a database table, a WORD-template, and an HTML/ASP document form, and that causes such to be displayed on-screen to a user with an Internet-client browser;

wherein user exposure to programming and scripting languages is minimized though an easy-to-use web interface, and content programmers can create the necessary components for automating documents, database tables, and HTML/ASP document forms without needing SQL, HTML, or ASP programming skills or experience, and further where such can be used to generate any merge fields needed in the creation of the WORD-file.

Fig. 1


100
106
user
107
user
108
user
104
Internet
109
user
110
user
112
user
118
NIC
102
120
ASP
122
IIS
124
Windows-NT
126
SRVANY
128
MS-Word
130
other apps
136
database
job queue
132
138
disk
134
140
NIC
142
114
144
NIC
148
Windows-NT
152
SRVANY
156
MS-Word
160
other apps
116
146
NIC
150
Windows-NT
154
SRVANY
158
MS-Word
162
other apps

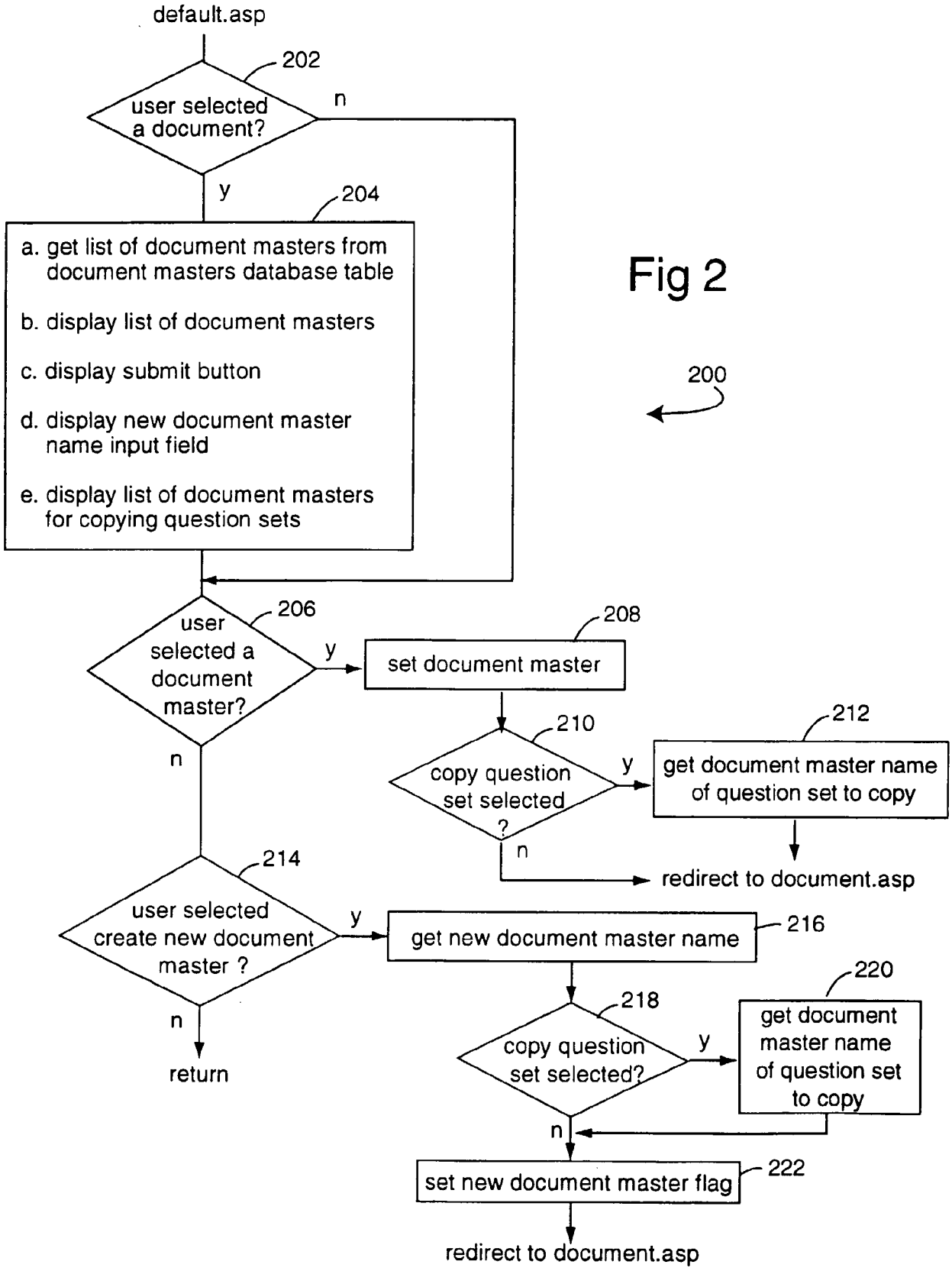

default.asp
202
user selected a document?
n
y
204
a. get list of document masters from document masters database table
b. display list of document masters
c. display submit button
d. display new document master name input field
e. display list of document masters for copying question sets
200
206
user selected a document master?
y
208
set document master
n
210
copy question set selected ?
y
212
get document master name of question set to copy
n
redirect to document.asp
214
user selected create new document master ?
y
get new document master name
216
n
return
218
copy question set selected?
y
220
get document master name of question set to copy
n
set new document master flag
222
redirect to document.asp

Fig 2

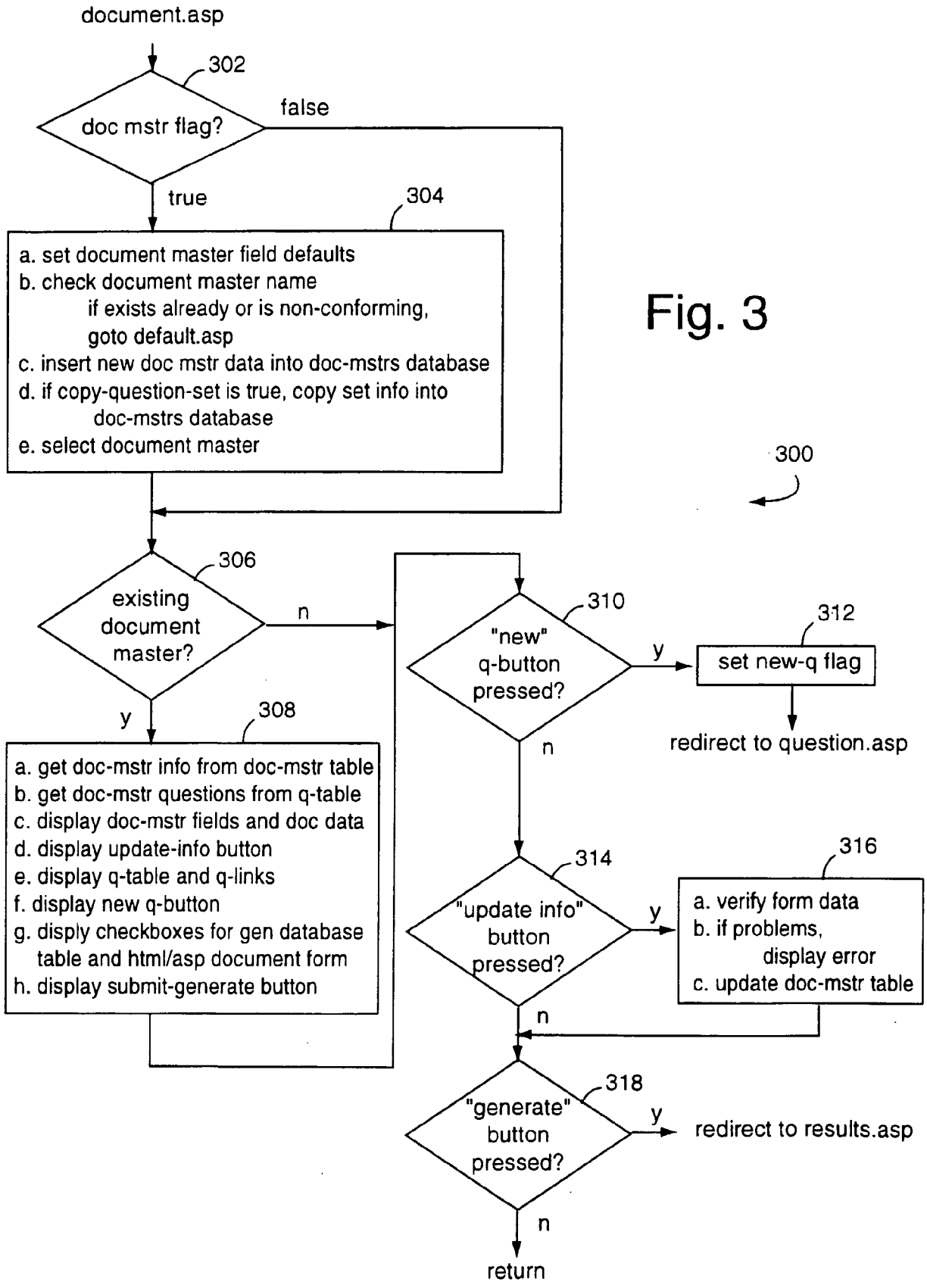

document.asp
302
doc mstr flag?
false
true
304
a. set document master field defaults
b. check document master name
if exists already or is non-conforming,
goto default.asp
c. insert new doc mstr data into doc-mstrs database
d. if copy-question-set is true, copy set info into
doc-mstrs database
e. select document master
300
306
existing document master?
n
y
308
a. get doc-mstr info from doc-mstr table
b. get doc-mstr questions from q-table
c. display doc-mstr fields and doc data
d. display update-info button
e. display q-table and q-links
f. display new q-button
g. disply checkboxes for gen database table and html/asp document form
h. display submit-generate button
310
"new" q-button pressed?
y
312
set new-q flag
redirect to question.asp
n
314
"update info" button pressed?
y
316
a. verify form data
b. if problems,
display error
c. update doc-mstr table
n
318
"generate" button pressed?
y
redirect to results.asp
n
return

Fig. 3

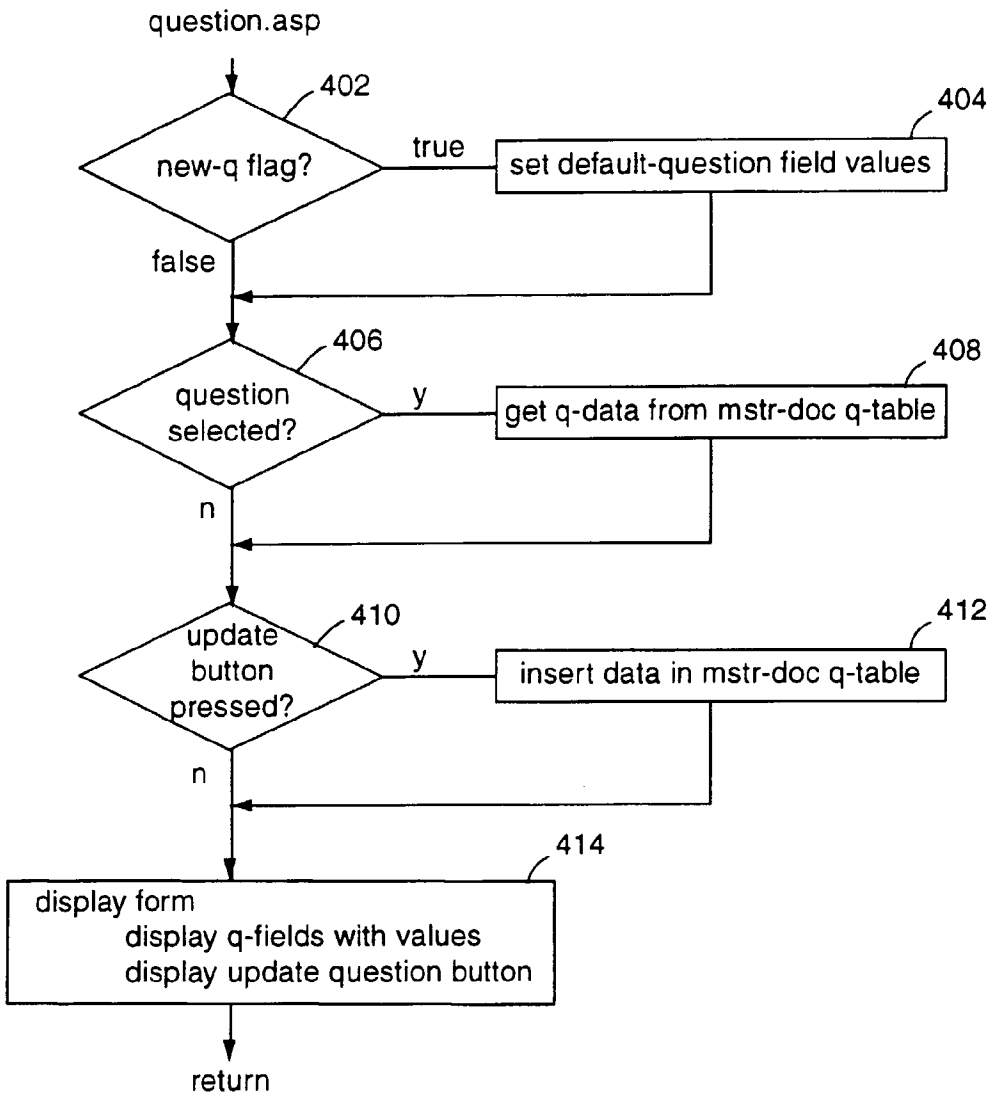

question.asp
402
new-q flag?
true
404
set default-question field values
false
406
question selected?
y
408
get q-data from mstr-doc q-table
n
410
update button pressed?
y
412
insert data in mstr-doc q-table
n
414
display form
display q-fields with values
display update question button
return
400

Fig. 4

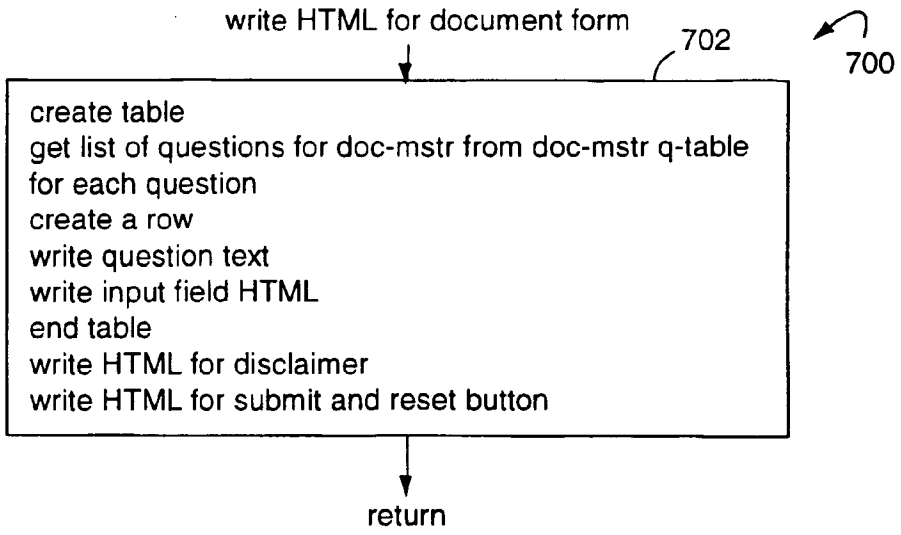

write HTML for document form
702
700
create table
get list of questions for doc-mstr from doc-mstr q-table
for each question
create a row
write question text
write input field HTML
end table
write HTML for disclaimer
write HTML for submit and reset button
return

Fig. 7

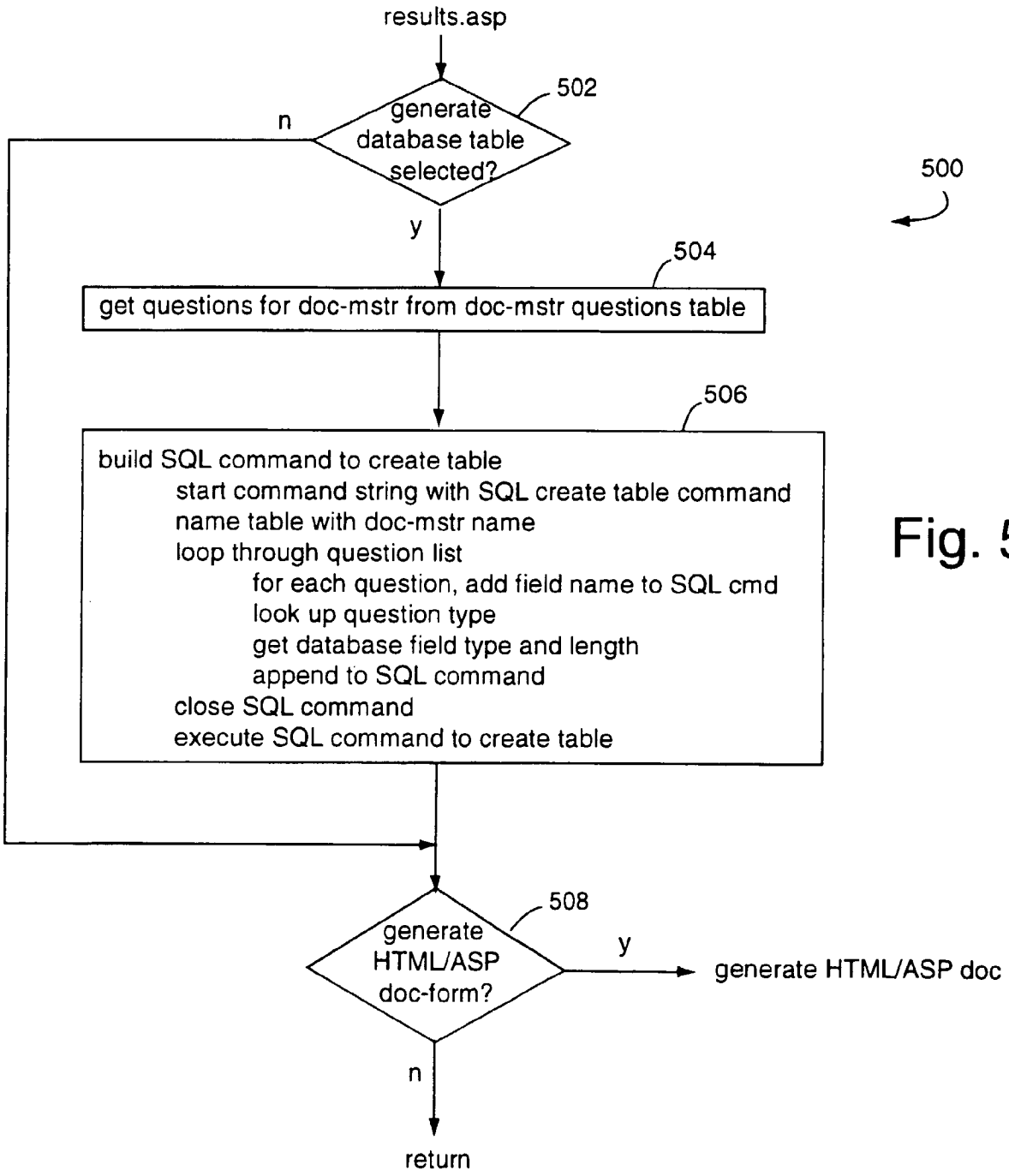

results.asp
502
generate database table selected?
n
y
500
504
get questions for doc-mstr from doc-mstr questions table
506
build SQL command to create table
start command string with SQL create table command
name table with doc-mstr name
loop through question list
for each question, add field name to SQL cmd
look up question type
get database field type and length
append to SQL command
close SQL command
execute SQL command to create table
508
generate HTML/ASP doc-form?
y
generate HTML/ASP doc
n
return

Fig. 5

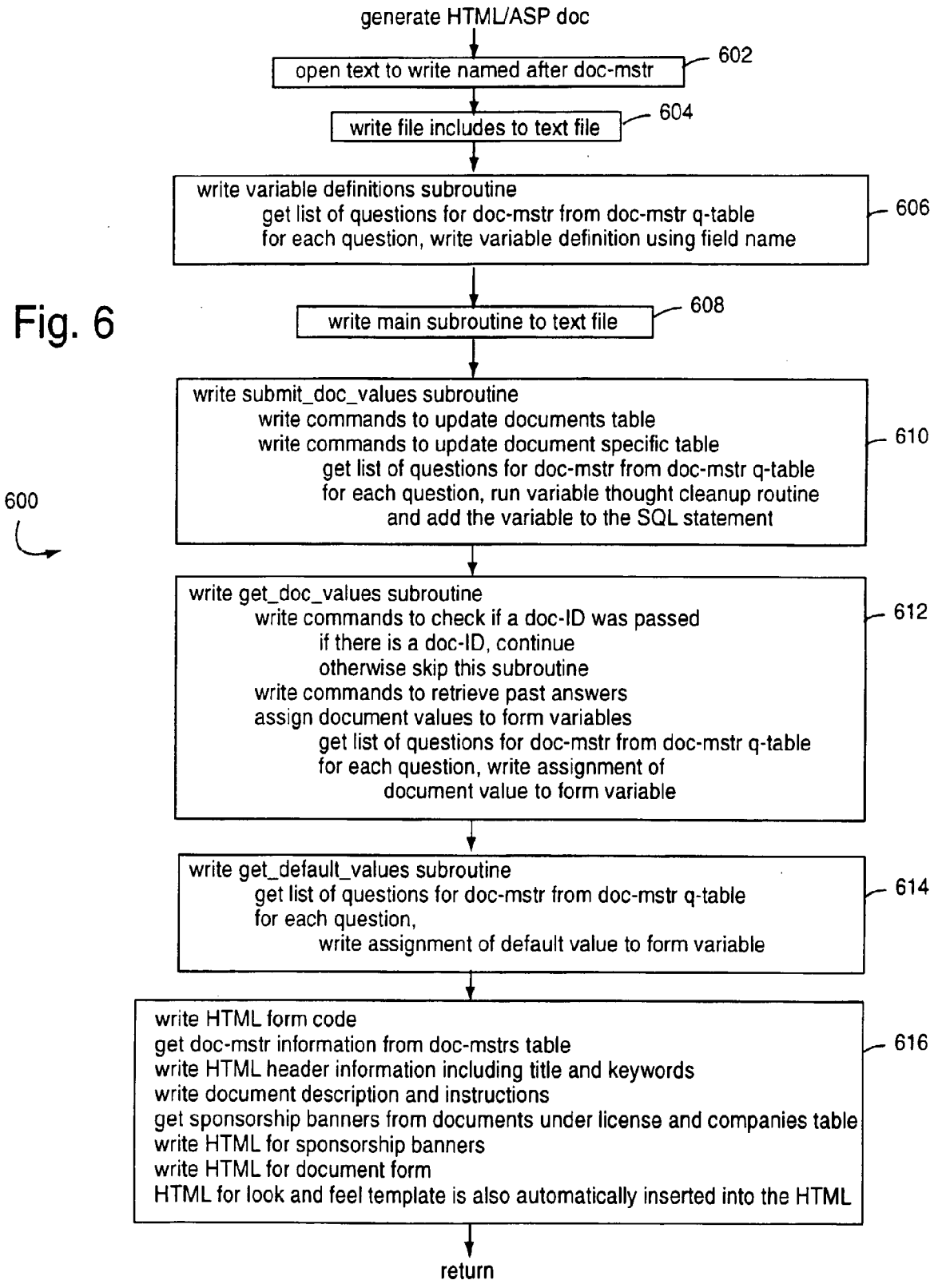

generate HTML/ASP doc
open text to write named after doc-mstr
602
write file includes to text file
604
write variable definitions subroutine
get list of questions for doc-mstr from doc-mstr q-table
for each question, write variable definition using field name
606
write main subroutine to text file
608
write submit_doc_values subroutine
write commands to update documents table
write commands to update document specific table
get list of questions for doc-mstr from doc-mstr q-table
for each question, run variable thought cleanup routine
and add the variable to the SQL statement
610
600
write get_doc_values subroutine
write commands to check if a doc-ID was passed
if there is a doc-ID, continue
otherwise skip this subroutine
write commands to retrieve past answers
assign document values to form variables
get list of questions for doc-mstr from doc-mstr q-table
for each question, write assignment of
document value to form variable
612
write get_default_values subroutine
get list of questions for doc-mstr from doc-mstr q-table
for each question,
write assignment of default value to form variable
614
write HTML form code
get doc-mstr information from doc-mstrs table
write HTML header information including title and keywords
write document description and instructions
get sponsorship banners from documents under license and companies table
write HTML for sponsorship banners
write HTML for document form
HTML for look and feel template is also automatically inserted into the HTML
616
return

Fig. 6

INTERNATIONAL SEARCH REPORT

IT onal application No.	PCT/US01/04872

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 7/00
US CL : 707/501, 513, 505-508, 530

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/501, 513, 505-508, 530

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
USPAT, USPG-PUB, DERWENT, EPO, JPO, IBM TBD

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,940,843 A (ZUCKNOVICH et al) 17 August 1999, column 2, lines 32-67, column 3, lines 1-67, column 4, lines 1-67	1-21
A	US 5,819,271 A (MAHONEY et al) 06 OCTOBER 1998, column 3, lines 45-67, column 4, lines 1-67, column 5, lines 1-67, column 6, lines 1-67, column 1, lines 1-67.	1-21
A	US 5,862,325 A (REED et al) 19 January 1999, column 8, lines 6-67, column 9, lines 1-67, column 10, lines 1-10	1-21
A	US 5,986,652 A (MEDL et al) 16 November 1999, column 3, lines 57-67, column 4, lines 1-20.	1-21
A	US 6,023,724 A (BHATIA et al) 08 February 2000, column 4, lines 36-67, column 5, lines 1-67, column 6, lines 1-67, column 7, lines 1-28.	1-21
A	US 6,026,433 A (D'ARALACH et al) 15 February 2000, column 2, lines 30-50.	1-21
A	HESTER, N., FRONTPAGE 2000 FOR WINDOWS: VISUAL QUICKSTART GUIDE, 1999, CH. 3.	1-21

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search 22 May 2001 (22.05.2001)	Date of mailing of the international search report 18 JUN 2001
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer HEATHER HERNDON Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998)